

**Exercice 1***Révisions sur les boucles*

1. Le script en Python ci-dessous doit répondre au problème suivant :

- prendre en entrée un entier  $p$ ;
- retourner le plus petit entier  $n$  tel que  $1^2 + 2^2 + 3^2 + \dots + (n-1)^2 + n^2 > 10^p$ .

Malheureusement ce script comporte plusieurs erreurs de syntaxe ou de conception.

Repérer et corriger ces erreurs.

---

```

1 p = input('Entrez un entier naturel p : ')
2 somme = 0
3 k = 0
4 seuil = 10**p
5 while somme>seuil:
6     somme = somme + k**2
7 print('le seuil de %s est dépassé à partir de n=%s'%(seuil,k))

```

---

2. Compléter le script Python ci-dessous qui prend en entrée une chaîne de caractères, la convertit en minuscules puis qui retourne une nouvelle chaîne où les voyelles ont été triplées.

---

```

1 chaine = input('Entrez une chaine de caractères :\n')
2 #on convertit la chaine en minuscules
3 chaine = chaine.lower()
4 voyelles = ['a','o','i','u','e','y','é','è','ê','ä','à','î','ù']
5 newchaine = ''
6 for lettre in chaine:
7     .....

```

---

```

1 Entrez une chaine de caractères :
2 Passer Noël sur une île au soleil.
3 Nouvelle chaine :
4 paaasseeer nooëëël suuur uuuneee îîleee aaauuu sooleeeiil.

```

---

exemple de sortie

**Exercice 2***Images et tableaux à deux dimensions*

1. On considère une image bitmap de dimensions (Largeur,Hauteur)=(800,600) c'est-à-dire que le tableau des pixels de l'image comporte 800 colonnes et 600 lignes. Les pixels sont codés avec le système de représentation (R,G,B) où R désigne la valeur de la composante Rouge, G la valeur de la composante Verte et B la valeur de la composante Bleue.

La profondeur de l'image est de 24 bits.

Si l'image n'est pas compressée, calculer le poids de l'image en mégaoctets puis en mébioctets. Arrondir les mesures à  $10^{-3}$  près. *On rappelle que 1 megaoctet  $\longleftrightarrow 10^6$  octets et que 1 mebioctet  $\longleftrightarrow 2^{20}$  octets.*

2. On considère des fichiers images dont la représentation bitmap est une matrice (ou tableau à deux dimensions) de pixels dont les valeurs sont 0 (pour la couleur noire) ou 1 (pour la couleur blanche).

On dispose du script ci-après) :

---

```

1 from PIL import Image
2 import numpy as np
3
4 #variable globale : dimension de la représentation d'un pixel
5 DOTPITCH = 10
6
7 def dimensions(t):
8     """Retourne les dimensions (nombre de lignes, nombre de colonnes)"""
9     return len(t), len(t[0])
10
11 def matrice_to_image(tab, nom='image.png', n=DOTPITCH):
12     """Convertit en image de profondeur 1 bit (pixel noir ou blanc)
13     de nline*ncol cases de coté n pixels
14     une matrice de pixels tab de dimensions (ligne, colonnes)=(nline, ncol)"""
15     nline, ncol = dimensions(tab)
16     newtab = np.zeros((n*nline, n*ncol))
17     for i in range(nline):
18         for j in range(ncol):
19             newtab[i*n : (i+1)*n, j*n : (j+1)*n] = tab[i][j]*np.ones((n, n))
20     im = Image.new('1', (n*ncol, n*nline)) #Image.new(mode, (Largeur, Hauteur))
21     im.putdata(newtab.flatten())
22     im.save(nom)
23     im.show()
24
25
26 def matrice_monochrome(H, L, valeur):
27     """Création d'une matrice de pixels de dimensions (nblignes, nbcolonnes)=(
28         hauteur, largeur) où tous les pixels ont la meme valeur """
29     t = []
30     for line in range(H):
31         newline = []
32         for col in range(L):
33             newline.append(valeur)
34     t.append(newline)
35     return t

```

---

### Script de départ

Ainsi l'image ci-contre (on a rajouté un cadre orange pour la délimiter) dont la matrice de pixels

est  $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$  est implémentée sous la forme d'une

liste de listes `[[1,0,1], [0,1,0], [1,0,1]]` et elle est enregistrée sur le disque par l'appel suivant (pour mieux visualiser on a choisi de représenter chacun de ses pixels par 50 pixels de l'écran) :

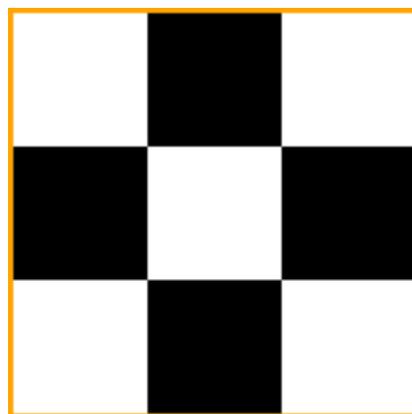
---

```

1 matrice_to_image
  ([[1,0,1], [0,1,0], [1,0,1]],
  nom='exemple.png', n=50)

```

---



- a.** Compléter le script avec une fonction `damier` qui retourne la matrice de pixels d'une image constituée d'un

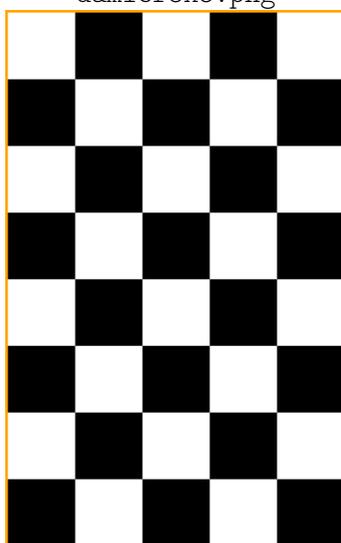
damier de case blanches noires (un pixel = une case). On donne en Annexe l'exemple l'image `damier8x5.png`<sup>1</sup> dont la matrice de pixels comporte 8 lignes et 5 colonnes.

Si ce n'est déjà fait, écrire une fonction `damier2` qui utilise la propriété suivante : dans un damier il n'existe que deux type de lignes.

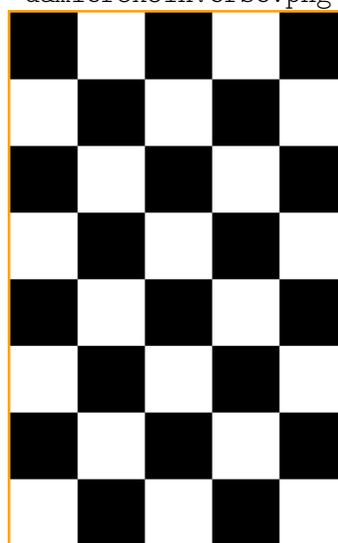
- b. Compléter le script avec une fonction `inverse` qui prend en paramètre une matrice de pixels et qui retourne une nouvelle matrice de pixels où tous les 1 ont été remplacés par des 0 et vice-versa. On donne en Annexe l'exemple de l'image `damier8x5inverse.png` dont la matrice de pixels a été obtenue à partir de celle de l'image `damier8x5.png` en appliquant la fonction `inverse`.
- c. Compléter le script avec une fonction `transpose` qui prend en paramètre une matrice de pixels et qui retourne une nouvelle matrice de pixels où les lignes de la nouvelle matrice sont les colonnes de la matrice initiale. On donne en Annexe l'exemple de l'image `damier5x8.png` dont la matrice de pixels a été obtenue à partir de celle de l'image `damier8x5.png` en appliquant la fonction `transpose`.

## ANNEXE

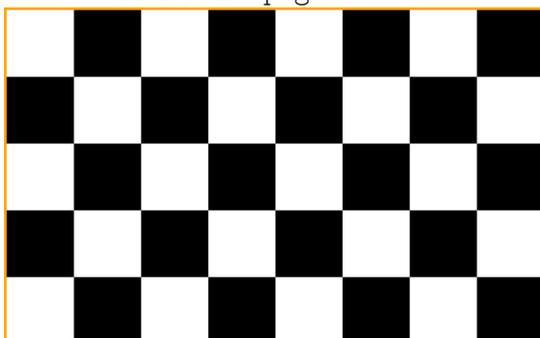
`damier8x5.png`



`damier8x5inverse.png`



`damier5x8.png`



1. on a rajouté un cadre orange à toutes les images pour les délimiter